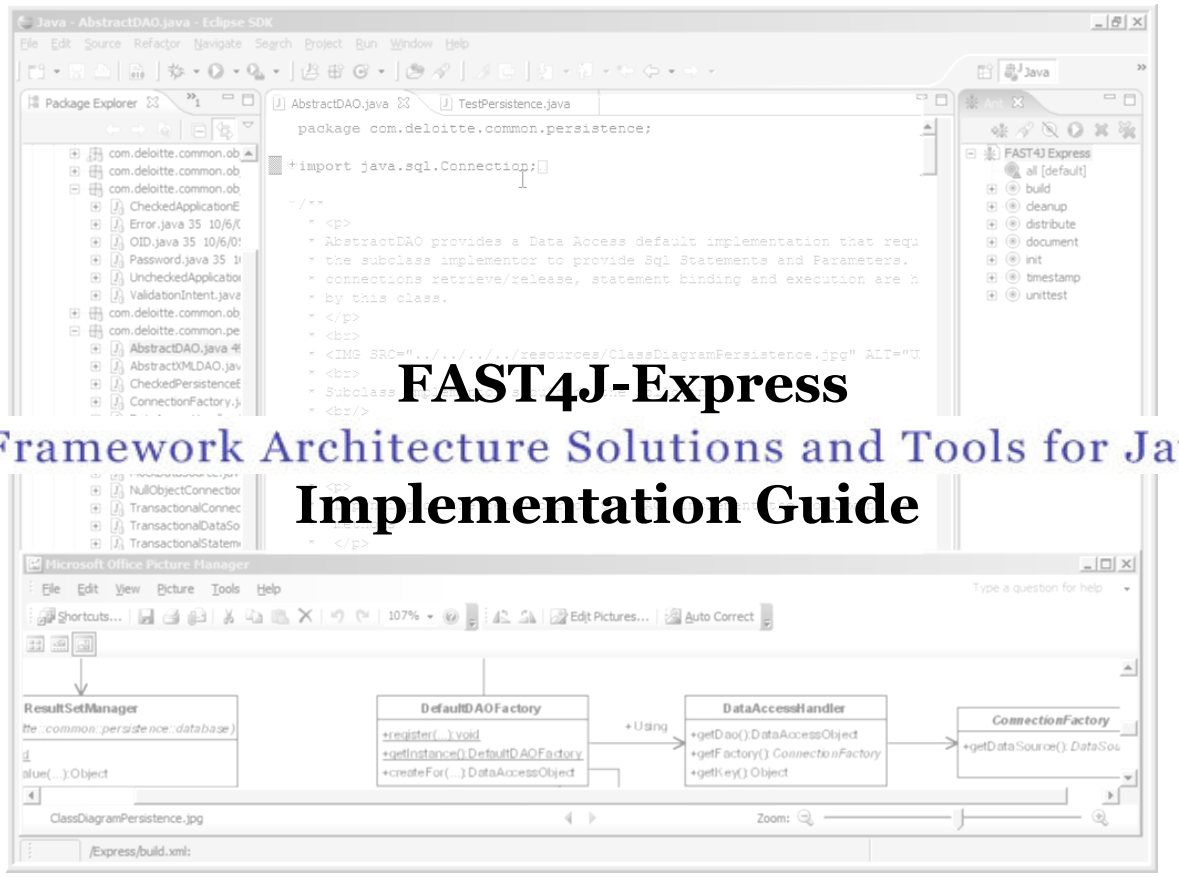


FAST4J-Express

Framework Architecture Solutions and Tools for Java

Implementation Guide

FAST4J Express Implementation Guide

Introduction

The FAST4J Express Framework is intended to speed up delivery of Custom Java Development projects. This guide should assist anyone in understanding the proper usage and implementation of the framework

The term “Express” will be used in this document to indicate the FAST4J Express Framework.

Intended Audience and Document usage

Anyone who is interested in understanding, using and delivering Custom Development solutions based on the Express Framework.

The document has been broken up into three main areas, Overview, Design and Usage.

Your Function	Document Area of Interest
Management	Framework Overview
Design and Architecture	Framework Design
Implementation	Framework Usage

Framework Overview

5 Minute Overview

The Express framework was built by Deloitte and is a collection of abstractions, patterns and design principles that together represent a J2EE Application Framework that can be built upon and extended. Parts of the framework can be used no matter what size Java project is being built. In its complete form, the Express framework is designed to be strong enough for small to medium sized Java projects, yet simple enough for most developers to understand and use in less than 1 day.

The 5 things you need to know about FAST4J Express

Cost	Free to the Project, free to the client. If you need additional support above and beyond the documentation, this may be available at low or no cost, depending on the level of support needed. If more robust support is necessary, you can build in the cost of the person to your project.
Licensing	There should be no licensing issues with using Express on any client engagement, as it is not dependent on any external software licenses. There are no issues such as 'viral licenses' with which to be concerned.
Feature List	Strong Domain Model Common Business Objects Persistence Services Layer Presentation Services
Complexity	The Express Framework strives to reduce complexity yet allow the development team to adapt the software to more complex solutions. It provides Interfaces and default implementations, that often just need one or two methods to be provided by the developer.
Support	If the default implementation does not meet the needs of the project, now or in the future, a feature request can be made to the framework team. In the meantime, the framework allows you to design and implement your own solution.

15 Minute Overview

Since its introduction, the Java solution set and specifically, J2EE has grown to be more complex than the problems it is meant to address. This complexity has led to confusion and conflicting "camps" in the industry. Often there is not just one right answer; as a result there are many ongoing debates, such as:

J2EE Servers versus lighter weight IoC containers
Entity Java Beans versus transparent persistence mechanisms
Distributed objects versus local calls
Remote Procedure Calls (RPC) versus Web Services

The core to the Express Framework is a solid architecture, well designed package structure, simple delivery mechanism and a built in set of features that can be used without modification if it matches the needs of the project, and in most cases it will.

Many frameworks are in the marketplace and each has its place. Here are some of the reasons why they might not meet the needs of Deloitte.

Vendor lock-in
: IBM, Oracle, BEA all have application frameworks, whereas each might be very good, they are not portable across environments, if not physically at least philosophically.

Licensing
: There are so many types of licenses (Apache, BSD, GPL, GNU, LGPL). Often clients will specifically disallow them in the implementations.

Complexity
: Many frameworks are more difficult to learn than the value they provide back to the developers or the project. Framework goals are often around scaling up, not scaling down, so there is much the project may not need.

Open Source Support
: Getting support on Open Source projects can seem difficult and knowing where to turn can be difficult to determine.

The cumulative effect of Complexity with the unpredictability of Open Source Support serves to increase project risk. At this point, the project team will typically make the decision to build for the functionality that might be available elsewhere.

How can the Express Framework overcome these problems for you?

We demand one simple thing from frameworks

$$TTL\text{-}F < PTS + CBNF + LF$$

The Time to Learn the Framework "must" be less than
Project Time Savings plus the
Cost Build New Functionality the framework provides and the
Loss of Flexibility.

Sometimes the equation has more of a qualitative aspect to it than quantitative. An example is the Loss of Flexibility. Just as in making an architectural decision; trade-offs are always necessary. Often it is too time consuming to elaborate every element of risk, so we make a best-guess estimate.

The Framework is first and foremost "easy" to use, and does not get in the way of the developers. The joints between the Framework and Application development code should be seamless, yet the Framework can be evolved and upgraded later if the project needs it.

Each and every element of functionality in the framework would have to be built into the application. As a result, there is no extra features and functionality. You will only utilize what you would have to build anyway. This means you have positive ROI, with the added benefit of having tested code at the beginning of your project.

Framework Design

Guiding Principles

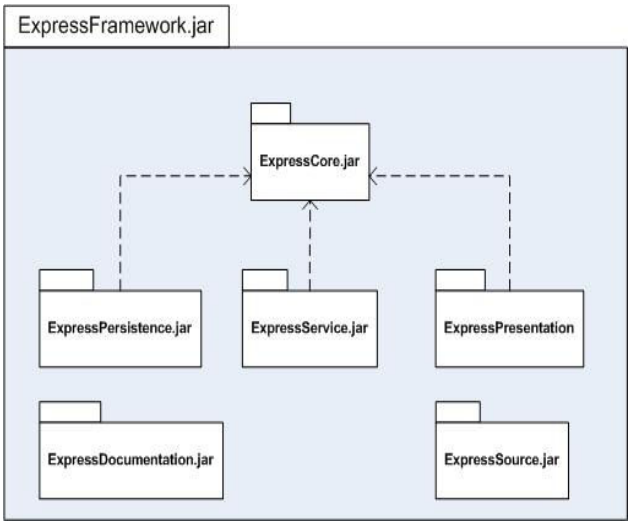
The focus of the FAST4J framework is not to replace superior products in the market-place. The following criteria should be met prior to the construction of framework features.

1. Simple to use and lightweight
2. No acceptable Open Source solution exists
3. Provide alternative solution to a Commercial Product
4. Firm believes feature will yield competitive advantage.

To facilitate an ala-carte deployment strategy, the Framework will be separated in such a way that it can be layered together to provide the full solution.

The Framework will be packaged in a series of Java Archive Resource(JAR) files that will each be self contained, with minimal dependencies.

The ExpressCore.jar is the base, to which all other jars should refer to. No circular dependencies should exist.



Express**Core**.jar
Core Interfaces, Domain Objects, Utilities and Common Business Objects

Express**Persistence**.jar
Persistence Interfaces, Persistence Commands, Connection Management, DAO's

Express**Service**.jar
Service Oriented Architecture Implementation

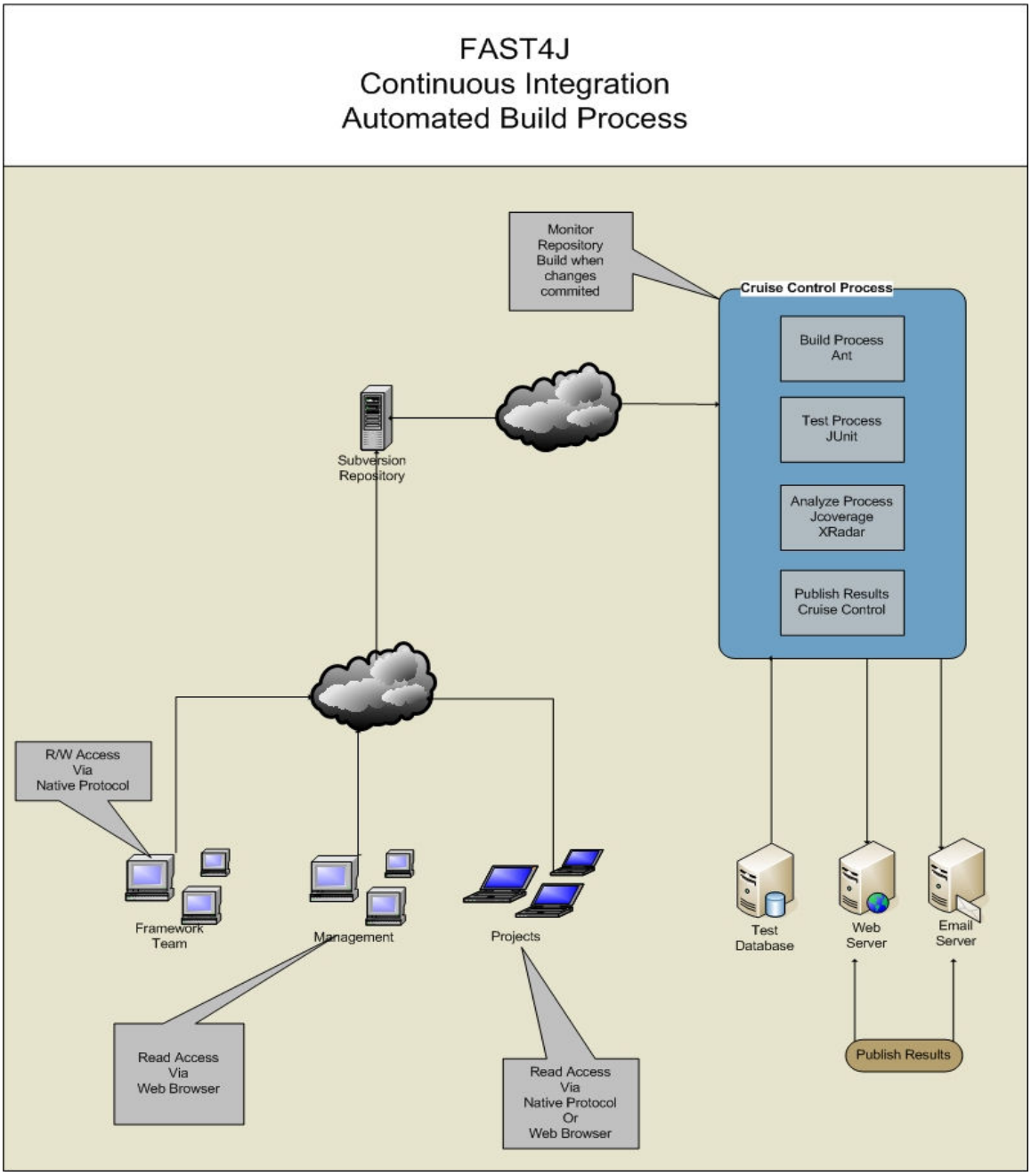
Express**Presentation**.jar
Web framework specific implementation

Express**Documentation**.jar
JavaDocs as well as this document

Express**Source**.jar
All source code

Continuous Integration

The Framework is under Subversion source control management. All source code and distributions are available in one location. The repository is monitored by CruiseControl, and when changes are committed, the automated build process is started. The automated build process will compile the changes, invoke the JUnits, perform a series of code analysis and coverage tools, and if all is successful package the application. The final step of the process is to publish the results via Email and a Web Server, so that anyone can determine the current status of the build.



Understanding Domain Objects

The key to a well-designed Object Oriented system is a strong Domain Model. To facilitate this one key aspect of the Express Framework is to strong domain objects.

The Framework defines Domain Objects as an Entity within an Object Oriented system that provides meaningful access to data and behavior. All useful Domain Objects have two things in common; they can be identified and they can tell you if they are valid.

Within the Express Framework, the following Interfaces have been identified

Validation	Returns a collection of errors based on a specific state
Identification	A unique identifier that can be used to reliably identify an object
DomainObject	Extends the Validation and Identification contracts
DomainAttribute	A business object that describes Domain Objects (eg. SSN)
SimpleObject	Objects that only requires Strings for value and description
UUID	Universally Unique Identifier

Understanding Interfaces, Default Implementations and Factories

The implementation strategy is based on three elements

Contract	Java Interface
Partial Functionality	Java Abstract Class
Implementation	Java Class that provides functionality

There are two popular strategies used to implement Object Oriented Frameworks; Interfaces and Abstract classes. Instead of forcing one versus the other, the Framework blends the strategies. A useful set of Interfaces have been defined, and where feasible an Abstract implementation provided.

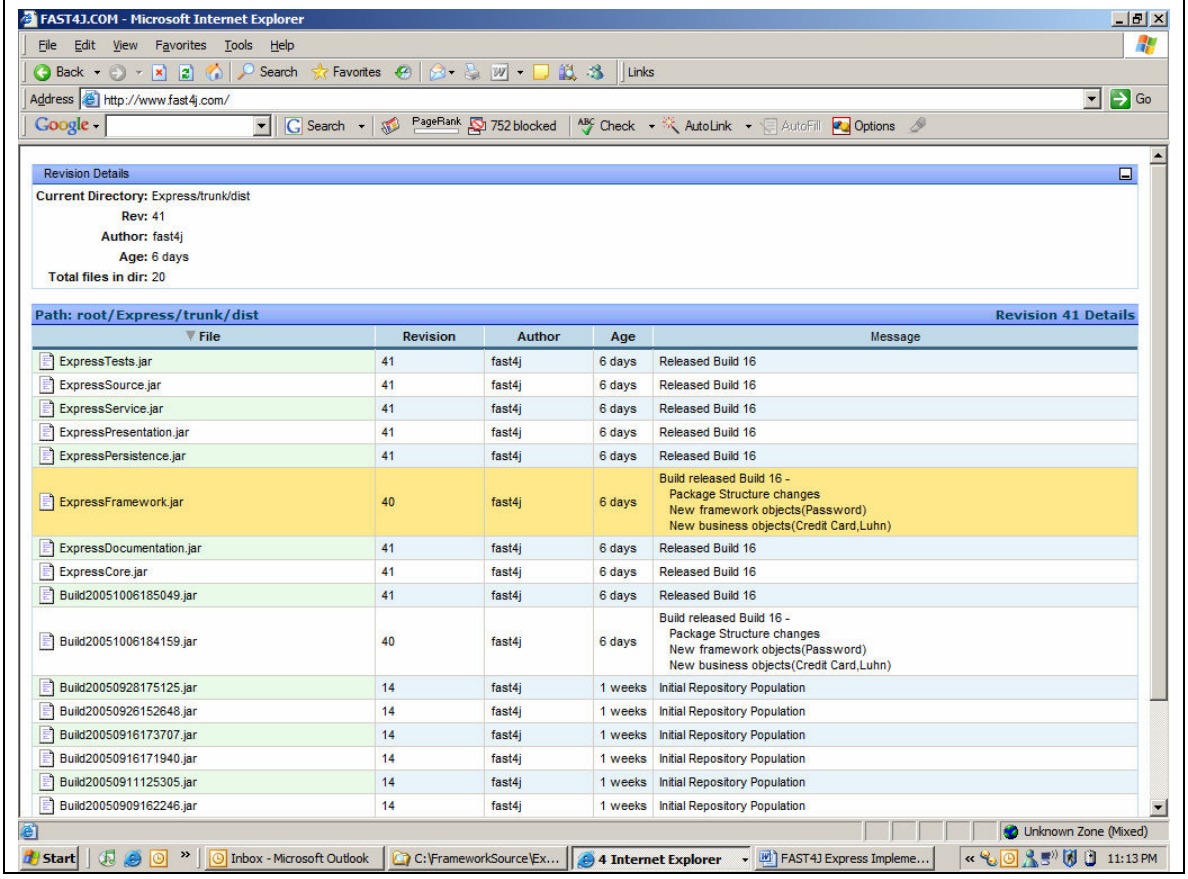
In many cases the Framework will also provide a Default implementation. This provides basic functionality that will meet the needs of most projects. If it does not meet the needs of the project, it will provide an excellent implementation example to be emulated.

Framework Usage

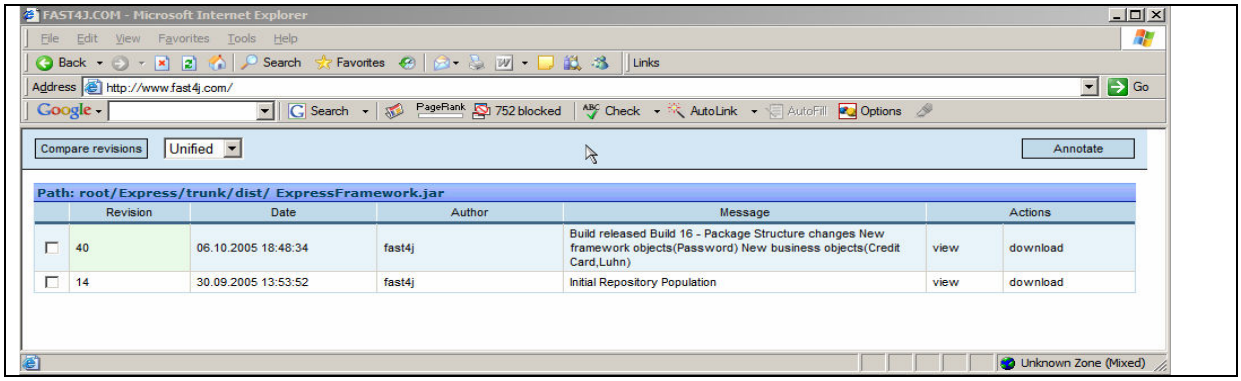
How to Begin

Getting the Framework

Go to the URL http://www.fast4j.com



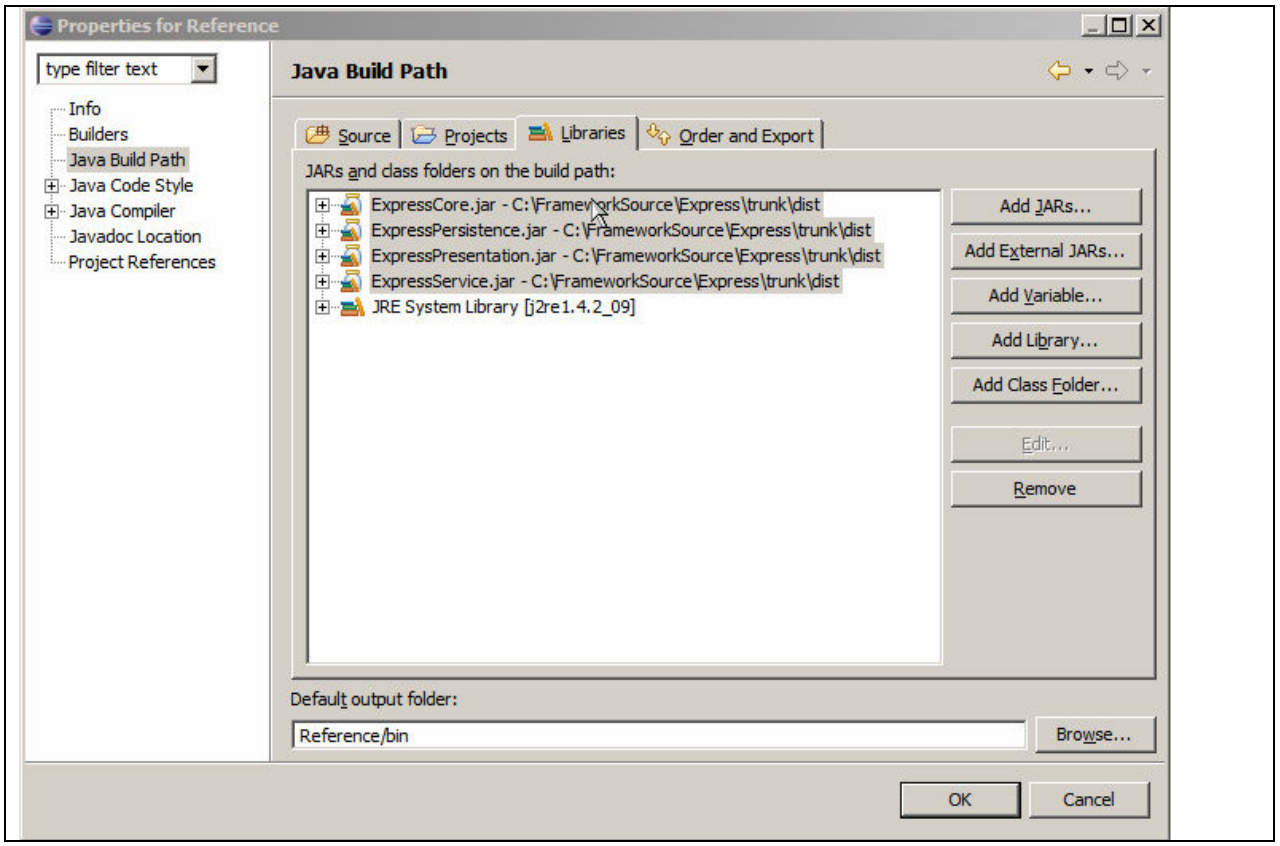
Click on the ExpressFramework.jar and download the Jar to your local machine.



Once you have downloaded ExpressFramework.jar file unpack it using WinZip or the Jar utility. Move the jars you need into the build path of your current project:

ExpressCore.jar
ExpressPersistence.jar
ExpressPresentation.jar
ExpressService.jar

Eclipse is used as an example to indicate how you might setup the project using an IDE.



Setting up your environment

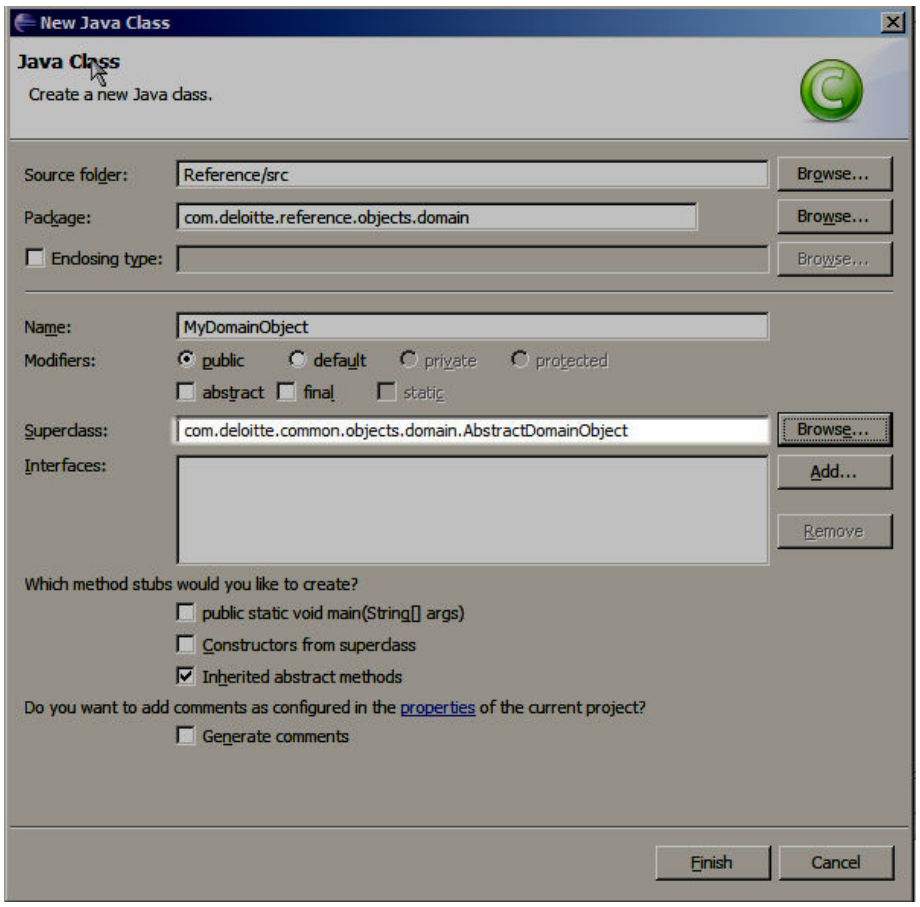
Implementing Interfaces

Key interfaces have been defined throughout the Framework. In almost every case the Application can provide the implementation that satisfies the Interface. Often behavior is not specified in the interface but is implied and understood by implementations. As a result, the Default implementation should be looked at for a basic understanding of what is implicitly expected by consumers of the Interface.

Note that extensive JavaDoc has been provided in the code, to facilitate this understanding.

Implementing Domain Objects

Domain objects are the fundamental building blocks of any Object Oriented system. They represent the "Entities" from OOA&D. To create one, the developer should extend the AbstractDomainObject class found in the com.deloitte.common.objects.domain package.



AbstractDomainObject by default provides identification behavior using the concept of an Object Identifier (OID), also known as a Universally Unique Identifier(UUID).

Once the AbstractDomainObject is extended, the developer can begin to add Application Business Rules that enable the object to provide functionality.

Business Rules are within the context of Intent, currently there are 5 Validation Intents that have been defined (None, Read, Update, Delete and Insert). With these intents most forces that act upon Domain Objects can be covered. Additional Intents can be created an implemented by the application. Typically this would be needed if the Domain Object has a known state that can be identified and supplied during validation.

Here is an example of a simple business object.

```
package com.deloitte.reference.objects.domain;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collection;

import com.deloitte.common.interfaces.DomainObject;
import com.deloitte.common.interfaces.Intent;
import com.deloitte.common.objects.domain.AbstractDomainObject;
import com.deloitte.common.objects.framework.Error;
import com.deloitte.common.objects.framework.ValidationIntent;

public class MyDomainObject extends AbstractDomainObject
{
   private Calendar startDate;
   private Calendar endDate;

   public MyDomainObject() { super(); }

   public MyDomainObject(UUID uuid) { super(uuid); }

   public Collection getValidationErrors(Intent intent)    {
                  Collection theErrors = new ArrayList();
                  if (startDate == null || endDate == null)
                  {
                        theErrors.add(new Error("Start/EndDate must be populated"));
                  }
                  else if (endDate.before(startDate))
                  {
                        theErrors.add(new Error("StartDate must be after EndDate"));
                  }
                  return theErrors;
   }

   public Calendar getEndDate() { return endDate;}
   public Calendar getStartDate() { return startDate;}
   public void setEndDate(Calendar endDate) { this.endDate = endDate;}
   public void setStartDate(Calendar startDate) { this.startDate = startDate;}

   public static void main(String[] args)
   {
         DomainObject theObject = new MyDomainObject();
         Collection theErrors = theObject.getValidationErrors(ValidationIntent.NONE);
         if (theErrors != null && !theErrors.isEmpty())
         {
                  System.err.println(theErrors);
         }
   }
}
```

MyDomainObject has two member variables, StartDate and EndDate. The only rule that we are implementing is that the End Date can not be before the Start Date.

Executing this class will yield the following results:

```
[Start/EndDate must be populated]
```

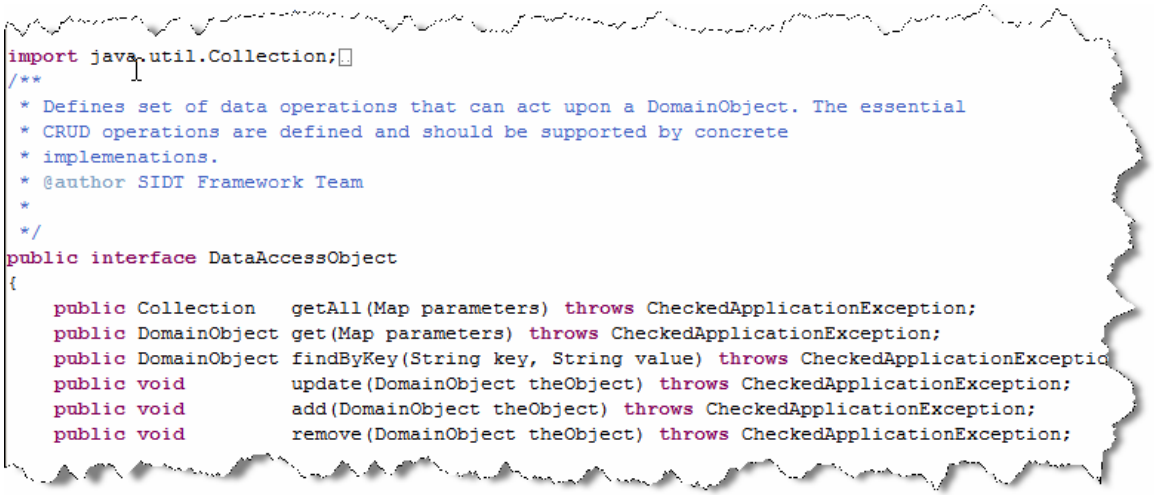
If we were to populate the Start Date with (01/01/2005) and the End Date with (12/31/2004), and re-execute we would get the following message:

```
[StartDate must be after EndDate]
```

Lastly, if we were to correctly set the End Date after the start Date, no error messages would be returned in the Collection, and processing could continue.

Hopefully this illustrates how the basic DomainObject can be used to build a larger more robust system. In the *Implementing Persistence* portion of this guide, additional examples will be provided as to how the DomainObjects can be acted upon by the Persistence Framework to be moved into and out of a repository such as a relational database.

The DataAccessObject interface is the primary entry point into the Persistence Layer.



```
import java.util.Collection;
/**
 * Defines set of data operations that can act upon a DomainObject. The essential
 * CRUD operations are defined and should be supported by concrete
 * implemenations.
 * @author SIDT Framework Team
 *
 */
public interface DataAccessObject
{
    public Collection   getAll(Map parameters) throws CheckedApplicationException;
    public DomainObject get(Map parameters) throws CheckedApplicationException;
    public DomainObject findByKey(String key, String value) throws CheckedApplicationExceptio
    public void         update(DomainObject theObject) throws CheckedApplicationException;
    public void         add(DomainObject theObject) throws CheckedApplicationException;
    public void         remove(DomainObject theObject) throws CheckedApplicationException;
```

The Framework includes an AbstractDAO to provide the bulk of the behavior necessary to implement Persistence. Connection and ResultSet handling will be provided and the sub-class should be focused on the mapping of Objects to the RDBMS.

The following implementation of a DAO is all that is needed to retrieve a DomainObject from the database.

In the specific example provided only a primary key (OID) will be used to access the object from the Database. The AbstractDAO will return the OID if it is found in the map provided to the DAO.

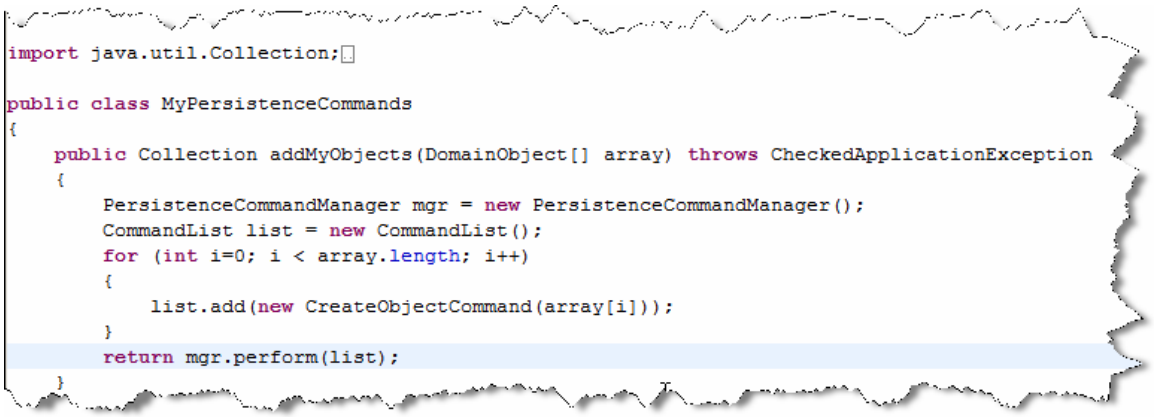
```
package com.deloitte.reference.persistence;

import java.util.Calendar;
import java.util.Map;

import com.deloitte.common.interfaces.DomainObject;
import com.deloitte.common.objects.framework.OID;
import com.deloitte.common.persistence.AbstractDAO;
import com.deloitte.reference.objects.domain.MyDomainObject;

public class MyDomainObjectDAO extends AbstractDAO
{
        protected DomainObject map(Map map){
                MyDomainObject theObject =
                        new MyDomainObject(new OID((String)map.get("OID")));
                theObject.setStartDate((Calendar)map.get("STARTDATE"));
                theObject.setEndDate((Calendar)map.get("ENDDATE"));
                return theObject;
        }
```

O/R Map Method

```
        protected String getSelectStatement(Map theParameters)
        {
           StringBuffer sql =
                 new StringBuffer("select oid,startdate,enddate from mytable where 1 = 1 ");
           if (theParameters.get("OID") != null )
           {
                 sql.append(" and oid = ? ");
           }
           return sql.toString();
        }
}
```

SQL Select Statement

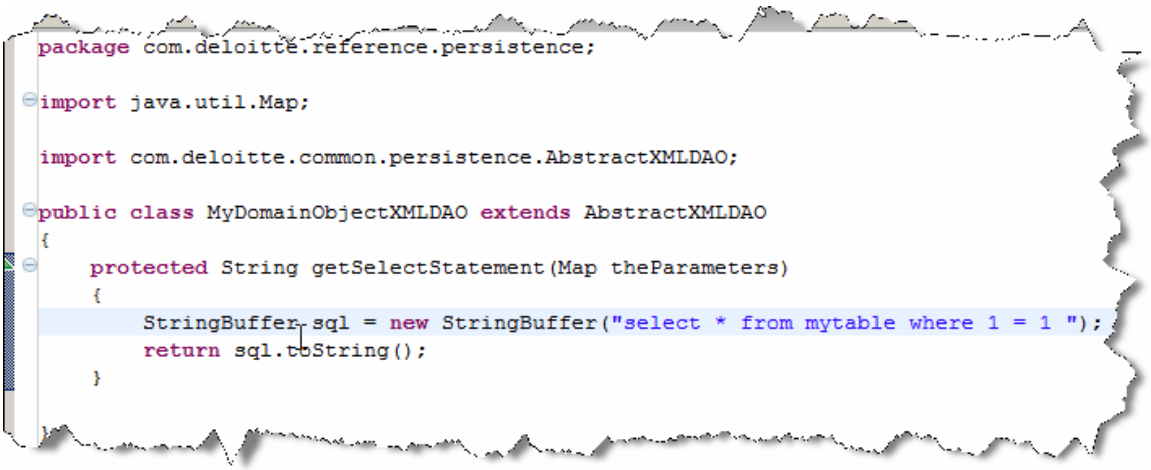
When performing transactional persistence commands one of the Framework's Persistence Commands (Create/Update/DeleteObjectCommand) should be used.



```
import java.util.Collection;

public class MyPersistenceCommands
{
    public Collection addMyObjects(DomainObject[] array) throws CheckedApplicationException
    {
        PersistenceCommandManager mgr = new PersistenceCommandManager();
        CommandList list = new CommandList();
        for (int i=0; i < array.length; i++)
        {
            list.add(new CreateObjectCommand(array[i]));
        }
        return mgr.perform(list);
    }
```

A set of Commands contained within a CommandList will be executed as a Transaction. Either all will fail, or all will succeed.

There are other DAO implementations provided, one important one to note is the AbstractXMLDAO implementation which provides the facility to retrieve a Database ResultSet as an XML Document. See the Extensions for how this might be useful.



```
package com.deloitte.reference.persistence;

import java.util.Map;

import com.deloitte.common.persistence.AbstractXMLDAO;

public class MyDomainObjectXMLDAO extends AbstractXMLDAO
{
    protected String getSelectStatement(Map theParameters)
    {
        StringBuffer sql = new StringBuffer("select * from mytable where 1 = 1 ");
        return sql.toString();
    }

}
```

Registering DataAccessObject Implementations

Each DAO should be registered with the DefaultDAOFactory if the AbstractDAO implementation is going to be used. This is a simple task, and can be done once for each JVM.

The example provided first tries to get an InitialContext, which would be there if running within a container, if it is not found, it tries the standalone database connection.

Here is an example Connection Factory

```
public class MyConnectionFactory extends ConnectionFactory
{
    public MyConnectionFactory()
    {
        super();
    }

    static
    {
        Context ctx = null;
        try {
            ctx = new InitialContext();
            if (ctx != null)
            {
                DataSource ds = (DataSource) ctx.lookup("jdbc/MyDataSource");
                setDataSource(MyConnectionFactory.class,ds);
            } else
            {
                org.hsqldb.jdbc.jdbcDataSource ds = new org.hsqldb.jdbc.jdbcDataSource();
                ds.setDatabase("jdbc:hsqldb:hsql://localhost/myDB");
                ds.setUser("Dave");
                ds.setPassword("Tiger");
                setDataSource(MyConnectionFactory.class,ds);
            }
        } catch (NamingException ne)
        {
            // Handle the error
        }
    }
```

And how to register

```
static
{
    boolean singleton = true;
    DefaultDAOFactory.register(new DataAccessHandler("Report1",new MyConnectionFactory(),new MyReport1(),singleton));
    DefaultDAOFactory.register("Report2",new MyConnectionFactory(),new MyReport2());
```

This static initializer was taken from MyPersistenceCommands as it would be the entry point into the Persistence Layer, this can go anywhere as long as the registration occurs before invoking a subclass of AbstractDAO.

The default registration is to create a new DAO each time one is retrieved from the factory as in MyReport1. If a singleton pattern is preferable, use the example provided for MyReport1.

Implementing Services

As many people now use the term "Web Service" interchangeably with Service Oriented Architecture, we will start with a definition of a Service.

"A Service is an implementation of a well-defined business functionality that operates independent of the state of any other Service defined within the system. Services have a well- defined set of interfaces and operate through a pre-defined contract between the client of the Service and the Service itself."

Samudra Gupta

Service Oriented Architecture is more a methodology than a concrete implementation of behavior.

The Framework provides several features to facilitate SOA as well as Web Services. One example is the Service Access Object (SAO). The AbstractSAO handles the underlying intricacies of retrieving a SOAP Document over Http much the way AbstractDAO shields the developer from the underlying details of JDBC.

The AbstractSAO will open and close the HttpUrlConnection respond to errors and return the XML Fragment from the SOAP envelope that the sub-class is interested in.

The DomainObject is defined in this case as a simple Quote that a Web Service will return.

```
package com.deloitte.reference.objects.domain;

import com.deloitte.common.objects.domain.AbstractDomainObject;

public class MyQuote extends AbstractDomainObject
{
    private String quote;

    public MyQuote() { super(); }

    public String getQuote()              { return quote; }
    public void   setQuote(String quote) { this.quote = quote; }
    public String toString()              { return this.quote;}

}
```

The URL and Protocol are the ServiceDescription, these are registered in the ServiceFactory. These are treated much the way DataSources are in the Persistence layer.

```
package com.deloitte.reference.service;

import com.deloitte.common.objects.service.ServiceDescription;
import com.deloitte.common.service.ServiceFactory;

public class MyServiceSource extends ServiceFactory
{
    public static String url = "http://webservices.codingtheweb.com/bin/qotd";
    public MyServiceSource()
    {
        super();
    }

    static
    {
        setServiceDescription(MyServiceSource.class,new ServiceDescription(url,"POST"));
    }
}
```

The last activity is to subclass the AbstractSAO and create the mapping between the service response and the DomainObject. This correlates to the DAO implementation from the Persistence Layer.

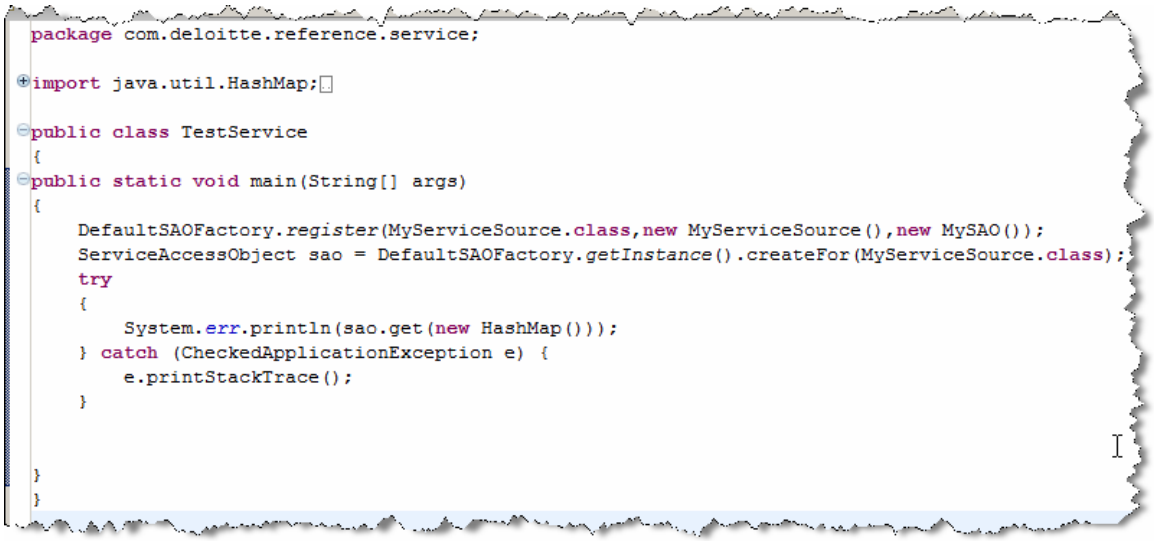
```
package com.deloitte.reference.service;

import java.util.Collection;

public class MySAO extends AbstractSAO
{
    protected final DomainObject map(Map map)
    {
        MyQuote theQuote = new MyQuote();
        theQuote.setQuote((String)map.get("RETURN"));
        return theQuote;
    }

    public Collection getAll(Map parameters) throws CheckedApplicationException
    {
        parameters.put("NODENAME","SOAP-ENV:Body");
        return super.getAll(parameters);
    }

    protected String getBody(Map parameters)
    {
        StringBuffer bodyXML = new StringBuffer();
        bodyXML.append("<SOAP-ENV:Body> ");
        bodyXML.append("<m:getQuote> ");
        bodyXML.append("</m:getQuote> ");
        bodyXML.append("</SOAP-ENV:Body>");
        return bodyXML.toString();
    }
}
```

Map values to Domain Object

Start of XML branch

SOAP Request Parameters

Test the Service



```
package com.deloitte.reference.service;

import java.util.HashMap;

public class TestService
{
public static void main(String[] args)
{
    DefaultSAOFactory.register(MyServiceSource.class,new MyServiceSource(),new MySAO());
    ServiceAccessObject sao = DefaultSAOFactory.getInstance().createFor(MyServiceSource.class);
    try
    {
        System.err.println(sao.get(new HashMap()));
    } catch (CheckedApplicationException e) {
        e.printStackTrace();
    }


}
}
```

Output from Execution

```
Everyone is a prisoner of his own experiences. No one can eliminate prejudices
- just recognize them. Edward R. Murrow (1908 - 1965)
```

Implementing Commands

Another strategy for implementing a Service Oriented Architecture is using the Command Pattern. An interface has been provided as well as several implementations.

```
package com.deloitte.common.interfaces;

import java.util.Collection;

import com.deloitte.common.objects.framework.CheckedApplicationException;
/**
 * Supports the Command pattern by seperating the invocation of
 * the command from the implementation.
 * <p>
 * Concrete classes that implement the Command interface will
 * consolidate behaviour that be invoked seperately or as a
 * collection of commands.
 *
 * @author SIDT Framework
 *
 */

public interface Command
{
    public Collection prepare() throws CheckedApplicationException;
    public Collection execute() throws CheckedApplicationException;
}
```

The PersistObjectAbstractCommand implements the Command Interface and allows for subclassing by the 3 Update Persistence Commands, here is one example.

```
package com.deloitte.common.commands.persistence;

import com.deloitte.common.interfaces.DomainObject;
/**
 * CreateObjectCommand accepts a DomainObject for <b>Creation</b>.
 * <p>
 * The getDAO(DomainObject theObject) method may be overriden to use a
 * different strategy for returning the DAO based on the DomainObject passed in.
 *
 * @author SIDT Framework Team
 */

public class CreateObjectCommand extends PersistObjectAbstractCommand
{

    public CreateObjectCommand(final DomainObject anObject)
    {
        super(anObject);
    }

    protected final void executeDAO() throws CheckedApplicationException
    {
        getDAO().add(theObject);
    }

    protected final Intent getIntent()
    {
        return ValidationIntent.INSERT;
    }

}
```

By instancing the CreateCommandObject using the DomainObject constructor and passing the Command into a CommandManager, the invoking class needs to know very little about the implementation details.

Another Command Manager has been provided which is the PersistenceCommandManager, this was mentioned in the Persistence Section of this document.

Tying it all together

After creating a MyQuoteDAO, we implement the Select and the Insert methods, the Select can be referenced in the Persistence Section, here is a snippet of the Insert method

```
protected String getInsertStatement(DomainObject theObject)
{
    return "insert into myquotes (oid, quote) values(?,?)";
}

protected List getInsertParameters(DomainObject o)
{
    MyQuote theObject = (MyQuote)o;
    List theParms = new ArrayList();
    theParms.add(theObject.getID());
    theParms.add(theObject.getQuote());
    return theParms;
}
```

In this Code example, we get 5 quotes from the WebService, create DomainObjects, add them to the CommandList and invoke the CommandManager to save them all to the database.

```
import java.util.ArrayList;

public class TestService
{
public static void main(String[] args)
{
    DefaultSAOFactory.register(MyServiceSource.class,new MyServiceSource(),new MySAO());
    DefaultDAOFactory.register(MyQuote.class,new MyConnectionFactory(),new MyQuoteDAO());

    ServiceAccessObject sao = DefaultSAOFactory.getInstance().createFor(MyServiceSource.class);
    try
    {
        List quotesFromService = new ArrayList();
        for (int i=0; i < 5; i++)
        {
            quotesFromService.add(sao.get(new HashMap()));
        }

        CommandList  theCommands = new CommandList();
        for (Iterator i=quotesFromService.iterator(); i.hasNext();)
        {
            theCommands.add(new CreateObjectCommand((DomainObject)i.next()));
        }
        CommandManager mgr = new CommandManager();
        mgr.perform(theCommands);

    } catch (CheckedApplicationException e)
    {
        e.printStackTrace();
    }


}
}
```

Querying the database, shows the following results

OID	QUOTE
11299099397430001	I grow more intense as I age.Florida Scott-Maxwell
11299099399230001	We cannot control the evil tongues of others; but a good life enables us to disregard them.Cato the Elder (234 BC - 149 BC)
11299099400940001	Promises that you make to yourself are often like the Japanese plum tree - they bear no fruit.Francis Marion (1732 - 1795)
11299099402740001	There is no excellent beauty that hath not some strangeness in the proportion.Sir Francis Bacon (1561 - 1626)
11299099404240001	There are 10^11 stars in the galaxy. That used to be a huge number. But it's only a hundred billion. It's less than the national d

Framework Extensions

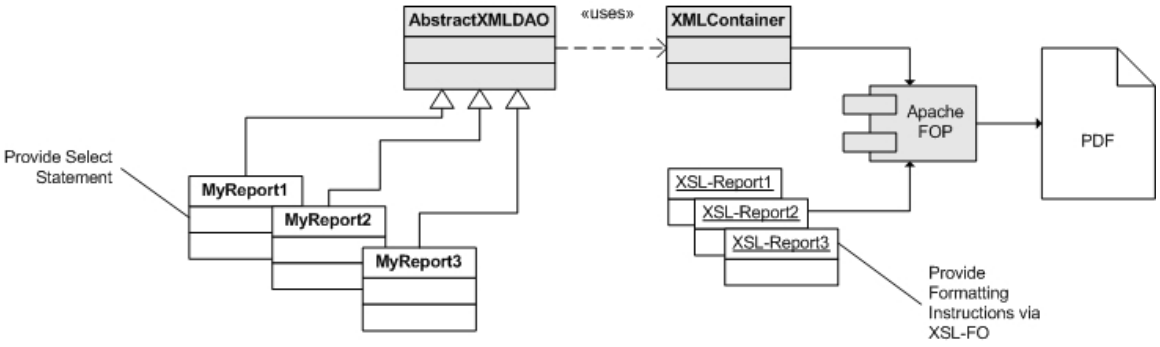
This section addresses a number of items although not directly supported by the Framework can be easily integrated based on several core features.

Dynamically generating PDF Documents from the Database

As part of W3C Specification for XSL is Formatting Objects (FO), which allows XML to be rendered into a user specified layout.

Several 3rd party products are available to facilitate this. One example; Apache-FOP, can be used to format XML into PDF.

Here is a sample design of how the Framework could be used to do this quickly and easily.



In each of the Report classes extending AbstractXMLDAO, the Select statements to product the ResultSet would be specified. The framework handles collecting the results and rendering it as XML. The XMLContainer object is a DomainObject that is returned from the DAO. At this point, applying the XSL-FO instructions is all that is needed to dynamically generate PDFs.

MyReport1.java XML DAO Implementation

```
package com.deloitte.extension.persistence;

import java.util.Map;

import com.deloitte.common.persistence.AbstractXMLDAO;

public class MyReport1 extends AbstractXMLDAO
{
    protected String getSelectStatement(Map theParameters)
    {
        StringBuffer sql = new StringBuffer("select region,sum(amount) from mytable group by region order by region ");
        return sql.toString();
    }
}
```

MyReportManager invokes Apache FOP Programmatically

```
package com.deloitte.extension.reports;

import java.io.*
import java.util.Collections;
import javax.xml.transform.*;
import javax.xml.transform.sax.SAXResult;
import javax.xml.transform.stream.StreamSource;
import org.apache.fop.apps.Driver;
import com.deloitte.common.objects.domain.XMLContainer;
import com.deloitte.common.objects.framework.CheckedApplicationException;
import com.deloitte.common.persistence.*;

public class MyReportManager
{
	private static MyReportManager me;
	private static File baseDir;
	private static File outDir;

	private MyReportManager() { }
	static
	{
		DefaultDAOFactory.register("Report1",new MyConnectionFactory(),new MyReport1());
		DefaultDAOFactory.register("Report2",new MyConnectionFactory(),new MyReport2());

		baseDir = new File(".");
		outDir = new File(baseDir, "out");
		outDir.mkdirs();
	}

	public static void generateReport1() throws CheckedApplicationException
	{
		generateReport("MyReport1",getXml("Report1"));
	}

	public static void generateReport2() throws CheckedApplicationException
	{
		generateReport("MyReport2",getXml("Report2"));
	}


	private static String getXml(String report) throws CheckedApplicationException
	{
		XMLContainer container =
(XMLContainer)DefaultDAOFactory.getInstance().createFor(report).get(Collections.EMPTY_MAP);
		return container.getXml();
	}

	private static void generateReport(String reportName,String input) throws CheckedApplicationException
	{
		File xsltfile = new File(baseDir, "reportformats/" + reportName +".fo");
		File pdffile = new File(outDir, reportName + ".pdf");
		convertXML2PDF(new ByteArrayInputStream(input.getBytes()), xsltfile, pdffile);
	}

	private static void convertXML2PDF(InputStream xml, File xslt, File pdf) throws CheckedApplicationException
   {
    Driver driver = new Driver();
     driver.setRenderer(Driver.RENDER_PDF);

     OutputStream out = null;
     try
```

```
    {
            out = new java.io.FileOutputStream(pdf);
        driver.setOutputStream(out);
        TransformerFactory factory = TransformerFactory.newInstance();
        Transformer transformer = factory.newTransformer(new StreamSource(xslt));
        Source src = new StreamSource(xml);
        Result res = new SAXResult(driver.getContentHandler());
        transformer.transform(src, res);
                    } catch (IOException e)
                    {
                            throw new CheckedApplicationException(MyReportManager.class,"Error during Report
Creation",e);
                    } catch (TransformerException e)
                    {
                            throw new CheckedApplicationException(MyReportManager.class,"Error during Report
Creation",e);

              } finally
              {
                try
                {
                        out.close();
                } catch (IOException ignoreThis) {}
              }
 }

            public static MyReportManager getInstance()
            {
                    if (me == null)
                    {
                            me = new MyReportManager();
                    }
                    return me;
            }

            public static void main(String[] args)
            {
                    try
                    {
                            MyReportManager.generateReport1();
                            MyReportManager.generateReport2();

                    } catch (CheckedApplicationException e)
                    {
                            e.printStackTrace();
                    }
            }
}
```

Stylesheet with XSL-FO Formatting Instructions

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:fo="http://www.w3.org/1999/XSL/Format"
        version="1.0">

<xsl:template match="ResultSet">
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

        <fo:layout-master-set>
           <fo:simple-page-master master-name="all"
                        page-height="11.5in" page-width="8.5in"
                        margin-top="1in" margin-bottom="1in"
                        margin-left="0.75in" margin-right="0.75in">
                           <fo:region-body margin-top="1in"
                                                margin-bottom="0.75in"/>
                           <fo:region-before extent="0.75in"/>
                           <fo:region-after extent="0.5in"/>
                </fo:simple-page-master>
        </fo:layout-master-set>

        <fo:page-sequence master-reference="all">
           <fo:flow flow-name="xsl-region-body">
                        <fo:block text-align="left">
                                Sales Totals By Region Report
                        </fo:block>
                        <fo:block text-align="left">
                                <fo:leader leader-pattern="dots"
                                        leader-length="2.5in"/>
                                <fo:table margin-top="5pt">
                                        <fo:table-column column-width="1in"/>
                                        <fo:table-column column-width="1in"/>
                                        <fo:table-body>
                                        <xsl:apply-templates select="Row"/>
                                        </fo:table-body>
                                </fo:table>
                        </fo:block>
                </fo:flow>
        </fo:page-sequence>
</fo:root>
</xsl:template>

<xsl:template match="Row">
        <fo:table-row>
                <fo:table-cell>
                        <fo:block font-size="12pt"
                                font-family="sans-serif">
                                <xsl:value-of select="REGION"/>
                        </fo:block>
                </fo:table-cell>
                <fo:table-cell>
                        <fo:block font-size="12pt"
                                        font-family="sans-serif">
                                <xsl:value-of select="AMOUNT"/>
                        </fo:block>
```

```
                </fo:table-cell>
        </fo:table-row>
</xsl:template>
</xsl:stylesheet>
```

Applied to the XML generated from the DAO

```
<ResultSet>
        <Row>
                <REGION>East</REGION>
                <AMOUNT>40000.00</AMOUNT>
        </Row>
        <Row>
                <REGION>North</REGION>
                <AMOUNT>12000.00</AMOUNT>
        </Row>
        <Row>
                <REGION>South</REGION>
                <AMOUNT>58000.00</AMOUNT>
        </Row>
        <Row>
                <REGION>West</REGION>
                <AMOUNT>60000.00</AMOUNT>
        </Row>
</ResultSet>
```

The PDF Report